Grid Data Management Pilot (GDMP): A Tool for Wide Area Replication in High Energy Physics - DRAFT *

Asad Samar
California Institute of Technology, Pasadena, USA
Asamar@cacr.caltech.edu

Heinz Stockinger
CERN, European Organization for Nuclear Research, Geneva, Switzerland
Institute for Computer Science and Business Informatics, University of Vienna, Austria
Heinz.Stockinger@cern.ch

October 12, 2000

Abstract

The CMS experiment at CERN, the European Organization for Nuclear Research, is currently setting up a Data Grid which will provide access to several Terabytes of data, distributed and replicated. In the real production environment data will be produced in different countries on both sides of the Atlantic by the end of year 2000. The stringent requirements of data consistency, security and high-speed transfer of huge amounts of data, imposed by the physics community need to be satisfied by an asynchronous replication mechanism. A pilot project called the Grid Data Management Pilot (GDMP) has been initiated which is responsible for asynchronously replicating large object-oriented data stores over the wide-area network to globally distributed sites. We present the design, architecture, functionality and performance results of our first working prototype. Different replication policies and protocols are supported that range from strictly synchronous to rather relaxed asychnronous models in terms of data consistency.

We believe that this first prototype can be regarded as a pioneer step towards a Data Grid and as a prototype for replication management within other Data Grid approaches like PPDG, GriPhyN and the EU-DataGrid[8, 14, 3].

1 Introduction

Next generation High Energy Physics applications are characterised by large amounts (several Petabytes) of mostly read-only data that are distributed and replicated around the globe. The visualisation and analysis of physics data will bring the physics community a step further in investigating the ultimate constituencies of matter, the big bang and hence our earth. Within the Compact Muon Solenoid (CMS) experiment at CERN, the European Organization for Nuclear Research, we are setting up a Data Grid [4] infrastructure required to fulfil the needs of the physics community. Note that this real word production has a smaller scale than the Data Grid project [10, 3] that starts officially in the beginning of year 2001.

Physics data in CMS are stored in object-oriented databases. Currently, existing commercial database management systems provide replication features but they fail to satisfy the stringent requirements of consistency, security and high-speed transfers of huge amounts of data, imposed by the physics community. An asynchronous replication mechanism that supports different levels of consistency, a uniform security policy and an efficient data transfer are necessary.

Recently Grids have become very popular in the distributed and parallel computing communities. Whereas in the past we have been speaking about meta and cluster computing, the trend has turned to Grid computing where the network between different nodes spans several countries and even continents. First data intensive application in the era of meta-computing were the so called Grand Challenges which dealt with weather forecast, climate simulation and aero- and fluid dynamics [16].

Basically existing Grid technology can be categorised into two major fields, the traditional computational Grids and data intensive Grids, called Data Grid. At CERN, we are currently setting up a Data Grid infrastructure that deals with several Petabytes of data distributed and replicated all around the globe [10]. Most of the data are read-only and require sophisticated data management strategies for distri-

*an extended abstract of the paper has been submitted for publication to: IASTED International Conference Applied Informatics (AI 2001) February 19-22, 2001 Innsbruck, Austria

bution, replication and access. One of the core middle-ware working groups of CERN's Data Grid project deals explicitly with these data management issues. Within the Compact Muon Solenoid (CMS) experiment at CERN, first data production tests are currently being done on the Grid. We expect several Terabytes of data to be produced and stored this year which will be a preparation for the Petabyte-scale data taking in the years 2005 and onwards. This pioneer step in the direction of a Data Grid requires a special software tool that deals with consistent, secure and high-speed transfers of huge amounts of data. The need for such a production system as well as the fact that the data management working group wanted to have a prototype project for assessing existing Grid technologies were the triggers for the Grid Data Management Pilot (GDMP) project. GDMP can be regarded as the first prototype of a Data Grid that is used in a production environment. It is currently being tested and used in the High Energy Physics (HEP) community, but the design is flexible and open enough to be applied to other data intensive research communities as well.

In the remainder of this paper we mainly address the specific requirements of the CMS experiment. Although most of the existing and next generation HEP experiments deal with large amounts of data, the data management is handled in different ways. This ranges from storing data in large flat files to deploying object-oriented and relational database management systems (DBMS). The CMS experiment has chosen an object-oriented software development environment, hence it is a natural outcome to use an object-oriented database for data storage. The DBMS of choice is Objectivity/DB [13].

A pilot project called Grid Data Management Pilot (GDMP) [9] has been initiated which is responsible for asynchronously replicating large object-oriented data stores over the wide-area network to globally distributed sites. We present the design, architecture, functionality and performance results of our first working prototype. The middle-ware of choice is the Globus [6] toolkit that provides promising functionality. We present test results which prove the ability of the Globus toolkit to be used as an underlying technology for a world-wide Data Grid. The required data management functionalities include high-speed file transfers, secure access to remote files, selection and synchronisation of replicas and managing the meta information. The whole system is expected to be flexible enough to incorporate site specific policies. The data management granularity is currently the file. However, future extensions of the system will also allow object replication.

We present a novel approach to use existing middle-ware technologies in order to handle the replication management of globally distributed data. Our first prototype is being used by the CMS experiment for the management of simulated physics data over the wide area on both sides of the Atlantic. We will be one of the pioneers to use the Data Grid functionality in a running production system. Although there is recently much effort going on in the Grid community to deal with the management of large amounts of data, the GDMP project can be viewed as an evaluator of different strategies, a test for the capabilities of middle-ware tools and a provider of basic Grid functionalities.

GDMP is a multi-threaded client-server system that efficiently and securely transfers files from one site to another one. It allows for certain network failure detection and recovery. Thus, an optimal utilisation of network bandwidth can be established. Sites that are temporarily unavailable via network links are themselves responsible for getting the latest information from other sites in the Grid. Different replication policies and protocols are supported that range from very stringent synchronous to rather relaxed asychnronous models in terms of data consistency [1, 5]. We adopt the Globus communication mechanism for messaging which is used by different components. Moreover, the responsibility of a single site to initiate data transfer can be changed. A subscription model allows a site to determine for itself the freshness of data. Even total independence and very infrequent requests for data can be established.

The current architecture and design presented in this paper is based on Objectivity. We make GDMP more flexible in the future to handle every kind of files, not just Objectivity files.

The paper is organised as follows. Section 2 elaborates on middle-ware requirements and the Globus features that are used by GDMP. Section 3 explains the data model, the role of the Objectivity and replication policies supported. The GDMP architecture is presented in the following section. Section 5 covers fault tolerance aspects and failure recovery mechanisms. In the next section we give our experimental results gained by transatlantic tests. Finally, we conclude the paper and give statements on future work.

2 Middle-ware Selection

2.1 Globus

The middle-ware is the basic infrastructure which acts as a uniform core for heterogeneous applications built on top. Several middle-ware tools exist for the Grid. Only to name a few of the most well known ones: Globus [6], Legion [12], Condor [2]. Based on some evaluation of these existing software systems we have chosen Globus as our core middle-ware toolkit since it provides the most commonly required functionality for a Grid infrastructure. Furthermore, Globus is also heading in the direction of Data Grids and is far more advanced in this aspect than any of the other existing middle-ware environments.

2.2 Features used from Globus

Since GDMP is currently only focusing on the Data Grid functionality, we use only a limited set of the entire Globus toolkit.

A main design issue in GDMP is to have a security component that enables authentication and authorisation. The Globus GSI security infrastructure provides the basic functionality we need.

In a distributed computing environment it is essential to have a communication mechanism which can be used by components to exchange control messages and to start services remotely. The Globus toolkit offers two distinctive communication libraries which we have evaluated by considering specific requirements for a replication system in a data intensive Grid application. These two libraries are Globus IO and Globus Nexus. Globus IO is a thin layer on top of basic socket communication and provides an API for easily managing sockets over TCP or UDP. Nexus on the other hand has a more abstract API that allows to handle functions and data types for message passing. Our comparison and discussions with the Globus team have lead us to use the Globus IO library. We use the Globus data conversion library for providing high level interfaces to our applications that are similar to RPC calls.

We use the Globus thread library for the multi-threaded GDMP server. This library implements a subset of POSIX thread standard and provides a portable Grid-enabled threaded environment.

Once the Globus Replica Catalogue and Grid-FTP are ready, we will include these into the GDMP architecture to allow for a more flexible file replication model. Currently an implementation of the WU-FTP server and the NC-FTP client are used for secure and fast file transfers and Objectivity's native catalogue is used for file management.

3 Architecture

The GDMP software consists of several modules that closely work together but are easily replaceable. In this section we describe the modules and the software architecture of GDMP. The core modules are Control Communication, Request Manager, Security, Database Manager and the Data Mover. An application which is visible as a command-line tool uses one or several of these modules.

3.1 Control Communication Module

We base our system on the client-server approach. This module takes care of the control communication between the clients and the servers. The module uses the Globus IO library as the middle-ware and builds high level functionality on top. It takes care of the intricacies related to socket communication over the wide area between nodes with heterogeneous architectures. The functionality includes starting and stopping the server, connecting and disconnecting the client to and from the server and sending and receiving messages at both the client and server ends. This module provides services to the modules in the layers above.

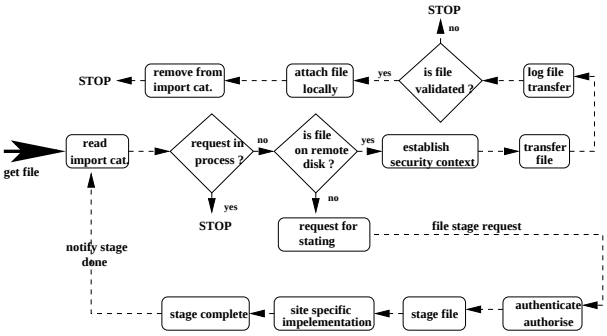

Figure 1. The actual file transfer

3.2 Data Mover Module

The main purpose of GDMP is to move files over the wide area in an automatic, efficient and fault tolerant way. This is the module which actually transfers files physically from one location to another one. It uses the NC-FTP client libraries to open a connection with the WU-FTP server on the server end. Since we use the GSI-NCFTP client and the GSI-WUFTP server, we have the same security mechanism (Globus Security Infrastructure) for both the Control Communication and the Data Mover. The functionality includes client authentication and authorisation before the transfer starts, transferring files to and from the server, validating the transferred files using the file size and checksum information, resuming the file transfer from the latest checkpoint after a network failure, using a progress meter to output the progress during a file transfer and finally logging all file transfers that have been completed. Our intention is to replace the NCFTP use with the upcoming Globus FTP libraries. This will enhance the performance of the Data Mover Module because of new features like partial file transfers, parallel file streaming and better fault recovery behaviours expected in the Globus FTP. Figure 1 gives the complete picture of the file transfer.

3.3 Security Module

Security is one of the main concerns in a Grid. Allowing people from outside one's domain to use the resources is a big issue for most organisations. Sensitivity of data and unauthorised use of network bandwidth to transfer huge files are the main security issues we are dealing with. This is

taken care by the Security Module. It is based on the Globus Security Infrastructure which is an implementation of the Generic Security Service (GSS) API. It uses the Public Key Infrastructure as the underlying mechanism. The Security Module provides methods to acquire credentials, initiate context establishment on the client side and accept context establishment requests on the server side (context is established to authenticate the client), encrypting and decrypting messages and client authorisation. The server authenticates and authorises any client before servicing its request, hence, the software protects a site from any un-wanted file transfers and blocks any un-authorised requests.

3.4 Request Manager Module

Every client-server system has to have a way to generate requests on the client side and interpret these requests on the server side. The Request Manager Module does exactly that. It contains several request generator/handler pairs and more can be added for customised use. This module is based on the Globus DC library which provides methods to convert data between different formats supported by variable machine architectures. The requests are generated on the client side by stuffing a buffer with the appropriate function handler, which is to be called on the server end, and any arguments required by that function. On the server side this buffer is unfolded and the data types are converted according to the local architecture. Finally, the respective function is called with the given arguments. The Request Manager Module basically mimics a limited Remote Procedure Call (RPC) functionality with the advantage of being lightweight and extensible.

3.5 Database Manager Module

This is the module which interacts with the actual Database Management System (DBMS). The module relies on the APIs provided by the particular data storage system being used. In our case the DBMS of choice is the Objectivity/DB, hence we use Objectivity's APIs to implement this module. To use GDMP in a different system, this is the only module which has to be swapped by the one which can interact with the specific DBMS. The functionality includes retrieving the database catalogue, containing information about the files currently present in the database, and attaching files to the DBMS once they arrive on the client side and are validated.

3.6 The GDMP Applications

GDMP includes some applications which are the customers of the services provided by the above outlined modules. These applications include various clients and one server.

3.6.1 The GDMP server

The GDMP server is a daemon constantly running on sites which produce data or want to export their data to other sites. We expect a large number of clients connecting to the GDMP server from all over the globe, transferring huge files which might take days to complete the transfer. Under such conditions the server is required to be very robust, extremely fault tolerant and able to cope with multiple clients simultaneously.

The server itself uses the communication module for receiving requests from application clients. Since thread creation is rather time consuming, the server uses a thread pool with a certain amount of threads which can be adapted. Each time a client is connecting, a thread is allocated to a single client. This is again a performance aspect, since the client and the server can communicate over one connection channel as long as the client is connected to the server. When the client disconnects, i.e. the application program terminates, the socket connection is closed and the thread is returned to the thread-pool.

For each client one thread is used. Thus, the number of threads in the server corresponds to the number of concurrent connections to the server. If no free thread is available, the client's request is put into a waiting queue. As soon as one thread terminates, the first request in the queue is served. The number of elements in the queue can be set on compilation time of the server.

Like a client application, also a server requires a proxy to be running. The server uses a dedicated server certificate which is included in the GDMP software distribution. Thus, when a server is started, the required Grid proxy is gained automatically. With the same account also GDMP client applications can be started but they need to get their own proxy based on the current user. Moreover, the server needs to have access to the local grid-mapfile of the site. The grid-mapfile can be given as an input parameter or a default location will be chosen on starting the server. Note that each GDMP application that request a service from a remote site has to authenticate at the server. Only if the client passes the authentication procedure, a connection to the server is established. Any output or error message from the server is logged in a corresponding file.

3.6.2 Client Application Programs

An application program is sending a request through the Request Manager to any other module that serves the user request. For instance, a user starts the tool `gdmp_replicate_file_get` which internally formulates a request that is generated by the Request Manager. This module uses the Globus Data Conversion library for transforming a request into a byte string for internal socket communication in the communication module. The commu-

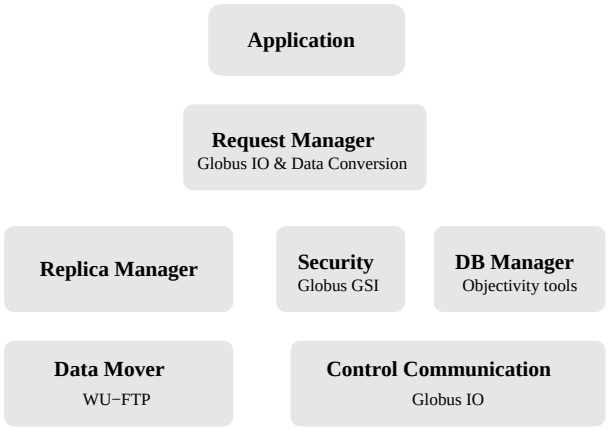

Figure 2. The GDMP Architecture.

nication module then communicates the required information to the server, which uses the Data Mover module to transfer the file from site A to site B. Once the file is transferred, it has to be integrated into the local Objectivity federation. Since this is a very database specific function, the DB-Manager takes care of vendor specific features for integrating files. Thus, the DB-Manager is very specific to the data storage requirements of the end-user and currently tightly coupled to Objectivity. However, the DB-Manager is extensible to any database management system or other storage systems like Root [15] which is used by several experiments in the HEP community.

Since all the access to data has to be done in a secure environment, a client has to be authorised and authenticated before he can request a service from the server. A client has to have a Globus proxy running in order to start the authentication process which is handled by the security module that is based on GSI, the Global Security Infrastructure. We use the *single login* procedure which is available through Globus, i.e. once a client has successfully got the proxy on one machine, he can send requests to any server without any further password entering (provided the local client is authorised to access the server).

Figure 2 shows the current architecture and all the modules of the software. On top of the architecture we have the Globus application, which can be multi-threaded. All the software modules are written in C++ and run on Solaris 2.6, 7 and Linux RedHat 6.1.

The clients provide command-line tools to the users, offering different functionalities. These basically act as the user interface to GDMP. Each client is customised to perform a specific task by communicating with the remote server(s).

4 Replication Policies and the Data Model

The GDMP software tool performs automatic, asynchronous replication of Objectivity database files in a Data Grid environment. Currently the software is restricted to replicate Objectivity/DB files only, but future extensions will allow to replicate files of any data type. The restriction to replicate only Objectivity files owes to the use of native Objectivity federation catalogue to handle files in GDMP. Once the Objectivity file catalogue is replaced by the announced Globus Replica Catalogue [7], a more flexible replication model can be supported.

4.1 Interfacing with the Data Production Software

In principle, a site, where Objectivity files are written, has to trigger the GDMP software which notifies all the ”subscriber” sites in the Grid about the new files. It is the responsibility of the data production software to trigger GDMP only after the Objectivity files have been completely written and are ready to be transported. In detail, the data production software at the local site uses the command-line tools provided by the GDMP software.

The ”subscriber” (destination) sites receive a list of all the new files available at the source site and can determine themselves when to start the actual data transfer. The data transfer is done with a WU-FTP server and an NC-FTP client. Since the usage of different machines in a Grid is a big security issue, a user has to be authenticated and authorised before contacting any remote site. The security is based on the GSI security toolkit available from Globus.

4.2 File Catalogues and a Subscription Model

We now elaborate how GDMP manages the transfer of files and their integration into the Objectivity catalogue. We illustrate the data flow by a producer-consumer example. The producer is the site where one or several Objectivity files are written, and the consumer is the site that wants to replicate these files locally. Once the producer has finished writing a set of files (or just a single file), it publishes this information by creating and sending an *export catalogue* to all the ”subscribed” consumers. The export catalogue has a listing of all the newly generated files and their related information. The consumer creates an *import catalogue* where it lists all the files that are published by the producer and have not yet been transfered to the consumer site. Figure 3 illustrates this model graphically.

The currently implemented replication policy is an asynchronous replication mechanism with a subscription model. A producer can choose at what time new files are written into the export catalogue and thus made publicly available for consumers. Hence, the producer can decide the

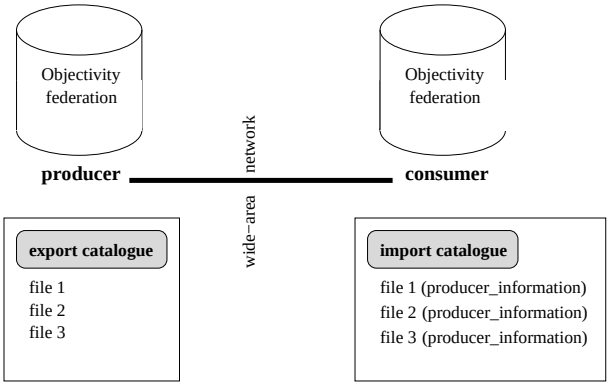

Figure 3. The role of the export and import catalogues

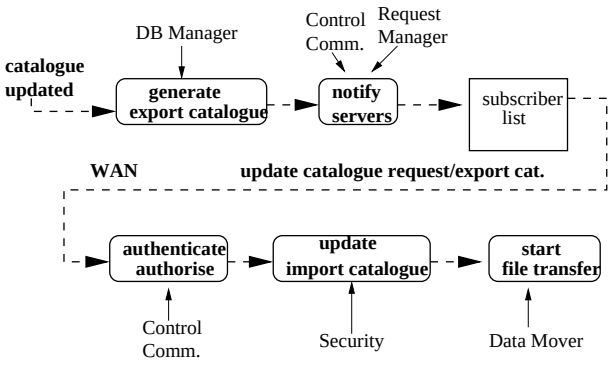

Figure 4. The control flow for a catalogue update.

degree of data consistency by delaying the publication of new files. In the example above we only have one consumer for demonstration purpose. In reality, the number of consumers can be infinite and depends on the amount of sites in the Grid. The subscription model enables that the subscribed consumers get informed immediately when new files are published in the export catalogue. Each consumer that wants be be notified about changes in the producer's export catalogue, subscribes to a producer. Depending on the degree of interest in a producer's data, consumers might want to subscribe to only some of the producers in the Grid.

Since the data exchanged has to be done in a controlled and secure way, a consumer first has to be "registered Grid user" at the producer site, i.e. the user has to be added to the grid-mapfiles of the producer site. These files contain all the users who are allowed to talk to GDMP servers running on a site. Thus a producer has total control over who subscribes to and transfers files from its site. Once this is done, a consumer is allowed to subscribe to the producer site. The producer then adds the new consumer and its related information in a file called *host-list*.

Once a producer has decided to publish the new entries made in the export catalogue (the tool `gdmp_publish_catalogue` is used), the producer sends the whole export catalogue to all the subscribed consumers. At the consumer site, the GDMP software creates the corresponding entries in local import catalogue of the consumer. The export catalogue can be regarded as an intermediate buffer that contains a list of newly created files. In more detail, the current Objectivity federation [1] catalogue and the old catalogue (the one which was available since the latest publication of the catalogue) are compared and the files which are new are inserted into the export catalogue. The export catalogue only contains newly created files and does not propagate information about deleting of files. This is regarded as a HEP-specific feature where files are not deleted but versions of old files are created.

The *export catalogue* contains the necessary information about the new files (host-name, port, filename with full directory path). The consumer can then decide when to start the file transfer from the producer to the consumer site with the tool `gdmp_replicate_file_get`. The tool reads the import catalogue and starts FTP sessions to get the necessary file. Once a file has safely arrived and is integrated into the local Objectivity federation, the file entry is deleted from the import catalogue. Section 5 describes what happens in case of broken connections and network failures.

Every transfer of a file, either successfully or not successfully done, is logged in a file called `replicate.log`. Figure 4 shows the control flow for a catalogue update.

4.3 Partial Replication: Filtering Files

GDMP allows a partial-replication model where not all the available files in a federation are replicated. In other words, one or several filter criteria can be applied to the import and/or export catalogue in order to sieve out certain files. For instance, a site only wants to make the files containing the word "Muon" in the filename, publicly available. Hence, the export catalogue, which contains all the files that a site wants to publish, has to be filtered and files that contain the filter criterium are deleted. A site that wants to get files from other sites can as well choose which files it wants to get. In this case the filter has to be applied on the import catalogue. This allows for a partial replication model where the producer as well as the consumer can limit the amount of files to be replicated.

One or several criteria can be created for the import as well as for the export catalogue with the tool `gdmp_filter_catalogue`. The citeria are stored in the files `etc/import_catalogue_filter` and

[1] A federation is the highest granularity of storing data in Objectivity. A federation consists of several database files which are managed by a federation catalogue.

etc/export_catalogue_filter.

4.4 The Role of a Site in the Data Grid

In our example above we distinguish between a producer and a consumer site in order to illustrate the data flow. However, each site in a Data Grid can produce new files and get files from other sites, i.e. a site can be producer and a consumer. Thus, each site manages an import catalogue where it stores available file information from other sites, and an export catalogue where is publishes files created by itself.

4.5 Dealing with Storage Limitations

Distributed sites may not have a tape subsystem. In other words, GDMP does not need a tape subsystem to work properly. In such a case, the HPSS and the CDR are left out and the ORCA production software directly interacts with the GDMP software by calling the GDMP clients.

A site may run out of storage space and hence some files have to be deleted when successfully replicated to the destination site. There is an option in `gdmp_get_catalogue` which allows to fetch the whole catalogue of a remote site. Based on the catalogue, a producer site can decide which files to delete. When `gdmp_get_catalogue` is used without a parameter, an import catalogue is created.

5 Fault tolerance and failure recovery

5.1 Site Failures

In a distributed system we can identify several sources for errors. On the one hand, there are hardware errors like a physically broken network cable, a broken network card, processor or any other piece of hardware. On the other hand, a part of the software system can have a failure. For instance, the FTP server dies, the file system crashes or some other part of the GDMP software does not work properly. In any of these cases the connection between the distributed clients and servers is broken. We refer to such a failure as the *connection is broken* and do not distinguish between the different reasons for this failure. We want to emphasise that the communication is broken and the message or control flow cannot be continued. For instance, in case of a broken connection anywhere between site A and site B, site A cannot publish its catalogue to site B.

In the current version of GDMP, each site is itself responsible for getting the latest information from any other site. This is also expressed by the subscription system, where a site has to explicitly subscribe to another site in order to get the file catalogue. Furthermore, when a site recognises a local error which has caused the broken connection, this site has to request from the peer site the required information. A site which publishes information to a subscribed site does not re-send information nor logs that a site could not receive the information. A site can retry to send the message again to the destination site within a particular time frame which can be set by a timeout parameter. If the re-sending fails again, the sending sites stops trying to contact the sites and hands over the responsibility to the destination site to recover from the broken connection.

To sum up, the policy is the following. Each site has to be aware of its state (connection okay or broken). Then it has to search for the origin of the broken connection. If it detects that the error is on the own site, is has to recover otherwise the peer site is responsible for failure recovery.

5.2 Recovery from Site Failures

A site can be unavailable for several hours or even days. Meanwhile several producers can have created and published files, and the entries in the export catalogues may already have been overwritten. Recall that a producer deletes the entries from the export catalogue once the catalogue has been successfully published to at least one consumer. A consumer can recover from the site failure by issuing the command `gdmp_get_catalogue`. Once a producer site has published its catalogue, the catalogue is available to be transferred to any consumer's site. GDMP at the consumer site then compares the consumer and producer catalogue and creates the necessary information in the consumer's import catalogue. Possible multiple appearances of files are deleted in order to keep the import catalogue's entries unique.

The process of successfully replication an Objectivity file from one site to another site consists of the following three steps:

1. transfer the file via FTP to the local site
2. attach the file to the local Objectivity federation via ooattachdb
3. delete the file from the import catalogue

This is also the order in which the GDMP software does the single replication steps. In the case of a broken connection, re-sending of several files is not needed since as soon as a file arrives safely at the destination site, the file is attached and the file entry is deleted from the import catalogue immediately. Only the file which is currently been sent when the network connection breaks, has to be resent. Since the implementation of WU-FTP has a "resume transfer" feature, not even the whole file has to be transfered but only the part of the file that is still missing since the last check point in the file. This allows for an optimal utilisation of the bandwidth in case of network errors.

Other possible errors: A site may publish a file several times and export it to other sites. In order to have files

only uniquely transferred, each site checks automatically if the file in the local import catalogue does not already appear in the federation catalogue. Furthermore, several sites can publish the same file to a specific site two times. Consequently, on creation of the import catalogue, the system checks if the file to be entered is unique. Only if this is the case, a new file is inserted into the import catalogue.

6 Experimental Results

Were are currently still working on some performance improvements and will put in the performance results by end October.

7 Conclusion and Future Work

We have been developing a data replication tool that allows for secure and fast data transfers over the wide-area network in a Data Grid environment. With our production ready software we have proven that Globus can be used as a middleware toolkit in a Data Grid. This has been a pioneer step in the direction of a Data Grid and to the best of our knowledge first software approach where a wide-area replication tool based on Globus is used in a production system. Furthermore, this work can also be regarded as an evaluator of Grid tools and thus has valuable input for other Data Grid activities like PPDG and GriPhyN.

The current architecture is restricted to Objectivity files but the system is kept flexible and extensible to include the announced Globus Replication Manager. Once this is integrated, we can provide a replication mechanism for any kind of files and the replica catalogue is managed by the Globus toolkit.

Moreover, in one of the next releases of GDMP we plan to include an object-level replication facility. This allows for a flexible asynchronous replication mechanisms for single Objectivity objects and supports end-user physics analysis on single workstations. Based on user access patterns to the Objectivity data store, local replicas will be created on demand in order to have faster access to data.

The current GDMP architecture only satisfies the needs for replication and does not include computational Grid aspects. We will extend the GDMP functionality by a job submission and scheduling task static load balancing. We will exploit the inherent parallelism of data production jobs in High Energy Physics.

Acknowledgements

We want to thank Tony Wildish (CERN) for initial work on a replication tool written in Perl. This tool and personal discussions have provided us with important input for the architecture of GDMP. Furthermore, thanks to Koen Holtman (Caltech) who is re-using and testing parts of our code and has pointed out some bugs in the software. Thank you also to Shahzad Muzaffar (Fermi National Lab.)for taking part in the transatlantic replication tests. Finally, we want to thank the Globus team, Luciano Barone (INFN), Dominique Boutigny (IN2P3), Johannes Gutleber (CERN), Mehnaz Hafeez (CERN), Wolfgnag Hoschek (CERN), Bob Jacobson (LBL), Werner Jank (CERN), Javier Jaen-Martinez (CERN), Veronique Lefebure (CERN), Harvey Newman (Caltech), Ben Segal (CERN), Ari Shoshani (LBL), and Kurt Stockinger (CERN) for their input and valuable discussions.

References

[1] Yuri Breitbart and Henry Korth, Replication and Consistency: Being Lazy Helps Sometimes, *Proc. 16 ACM Sigact/Sigmod Symposium on the Principles of Database Systems*, Tucson, AZ, 1997.

[2] The Condor Project Homepage: http://www.cs.wisc.edu/condor/

[3] The CERN DataGrid Project: http://www.cern.ch/grid/

[4] Ian Foster and Carl Kesselman (editors), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999.

[5] Jim Gray, Pat Holland, Patrick O'Neil, Dennis Shasha, The Dangers of Replication and a Solution, *SIGMOD*, 1996.

[6] The Globus Project: http://www.globus.org

[7] The Globus Data Grid effort: http://www.globus.org/datagrid/

[8] The GriPhyN Project, http://griphyn.org

[9] Mehnaz Hafeez, Asad Samar, Heinz Stockinger, A DataGrid Prototype for Distributed Data Production in CMS, to appear in *VII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT2000)*, October 2000.

[10] Wolfgang Hoschek, Javier Jaen-Martinez, Asad Samar, Heinz Stockinger, Kurt Stockinger, Data Management in an International Data Grid Project, to appear in *1st IEEE, ACM International Workshop on Grid Computing (Grid'2000)*, Bangalore, India, Dec. 2000.

[11] High-Performance Storage System (HPSS), http://www.sdsc.edu/hpss/

[12] Legion, A Worldwide Virtual Computer, http://www.cs.virginia.edu/ legion/

[13] Objectivity Inc., http://www.objectivity.com

[14] The Particle Physics Data Grid (PPDG), http://www.ppdg.net

[15] Root, An object-orient data analysis framework: http://root.cern.ch

[16] Heinz Stockinger. Dictionary of Parallel Input/Output, Master's Thesis. Department of Data Engineering, University of Vienna, Austria, February 1998.